SOFTVERSKO INŽINJERSTVO

NAPREDNE TEHNIKE PROGRAMIRANJA

nevzudin.buzadjija@unze.ba

- **15 sedmica, zimski semestar**

- **Predavanja:** **3x3 časa sedmično**
- **Profesor:**
  - prof.dr. Nevzudin Buzađija,
- **Asistent:**
  - V.as. Edin Tabak, MA – As. Narcisa Hadžajlić
- **Konsultacije:**
  - Ponedjeljak, 11,30 – 12,00
- **Literatura:**
  - Google Classroom
  - Absolute C++, 5th Ed., W. Sawitch, 2013.
  - Buzađija N. Čeke D (2021). Zbirka zadataka iz C++ sa elementima teorije
  - The C++ programming language, B. Stroustrup, 2013.

Sadržaj predmeta

Sedmica	Tema
I	01 Rekurzivne funkcije
II	02 Strukture
III	03 Pokazivaci
IV	04 Pokazivaci na funkcije
V	05 Dinamicka alokacija
VI	06 Manipulisanje nizom karaktera
VII	Test 1
XIII	07 Lamda funkcije
IX	08 Enumeracije
X	09 Funkcije clanice strukture
XI	10 Manipulisanje fajlovima
XII	11 Genericke funkcije i strukture
XIII	Uvod u objektno-orjentisano programiranje
XIV	Test 2

Ocjenjivanje: AR (SI)

Prisustvo predavanjima/vježbama	5+5
V – testovi sa vježbi, zadaci	
P – testovi sa predavanja, teorija (T1 ; T2+Z+DZ)	35+35+20 projekat

Predavanja	Vježbe	Praktičan rad	Teoretski ispit	Projektni zadatak	Domaća zadaća
10%	10%	30%	30%	10%	10%

- **Alternativno (umjesto T1, T2, T3)**
  - ZI÷I – završni ispit .....(40 bodova)

Svaki test (provjera znanja) > 51% !!!

- **Konačna ocjena (P+V+T+ZI) ili (P+V+ZI÷I)**

Bodovi	0÷54	55÷64	65÷74	75÷84	85÷94	95÷100
Ocjena	**5**	**6**	**7**	**8**	**9**	**10**

Periodični testovi (P) i ispit

	T1	T2
Datum	17.4.2023	12.06.2023
Dan	Ponedjeljak	Ponedjeljak
Vrijeme	9:00	9:00
Učionica		

Funkcije

Zadatak: $y=x^3$

Želimo izračunati $y=x^3$ pri čemu $x=2$, $x=3$.
Rezultat se ispisuje na zaslonu.

Klasično rješenje

```
#include<iostream>
using namespace std;
int main() {
  int x=2;
  int y=x*x*x;
  cout<<"2 na trecu je "<<y
        <<endl;
  x=3;
  y=x*x*x;
  cout<<"3 na trecu je "<<y
        <<endl;
  system("pause");
  return 0;
}
```

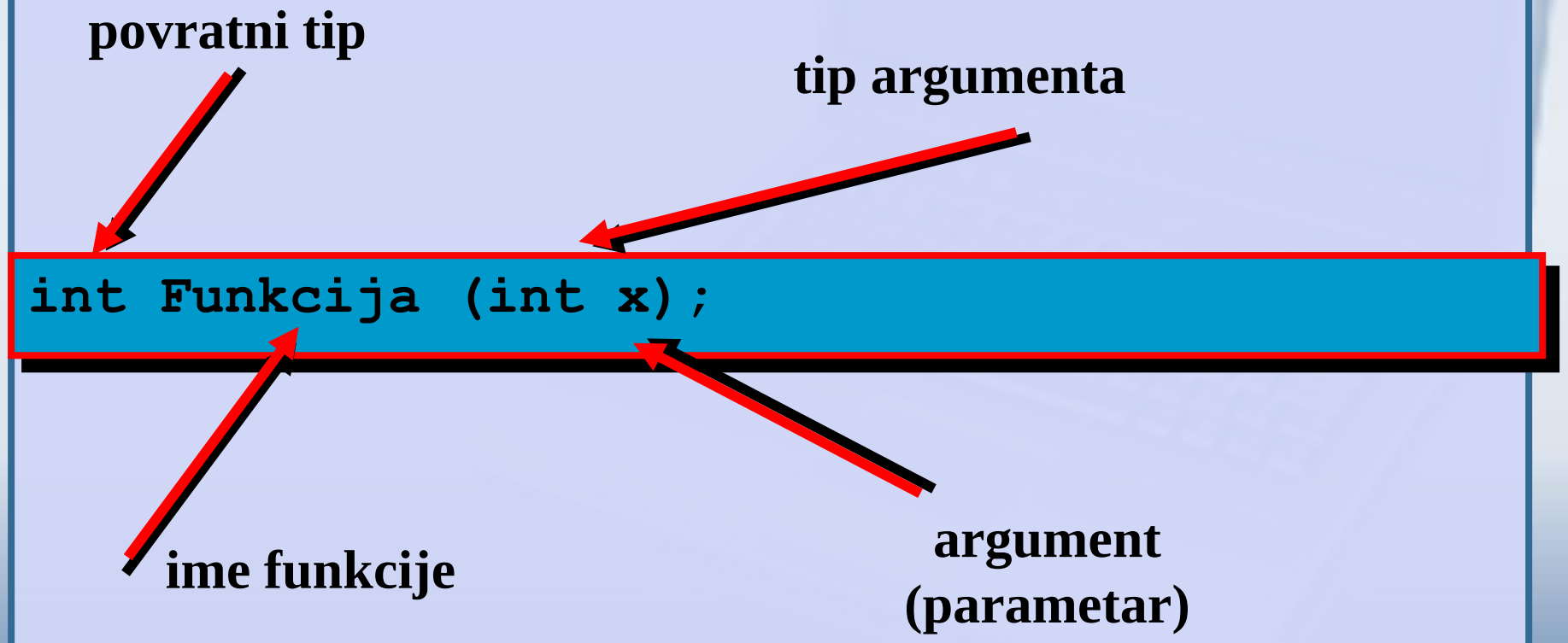
Rješenje s funkcijom

```
#include<iostream>
using namespace std;
int Funkcija(int x);
int main() {
    cout<<"2 na trecu je "
        <<Funkcija(2)<<endl;
    cout<<"3 na trecu je "
        <<Funkcija(3)<<endl;
    cout<<endl<<"Za kraj <1>: ";
    system("pause");
    return 0;
}
int Funkcija(int x) {
    return x*x*x;
}
```

deklaracija funkcije
poziv funkcije
definicija funkcije

Deklaracija funkcije: format

<povratni tip> ime_funkcije (<tip> argument_1, <tip> argument_2, ..., <tip> argument_n);



Funkcije: deklaracija - poziv - definicija

Deklaracija

prototip funkcije

```
int Funkcija (int x);
```

formalni argument

stvarni argument

Poziv

```
cout<< ... <<Funkcija(2)<<endl;
cout<< ... <<Funkcija(3)<<endl;
```

Definicija

formalni argument

```
int Funkcija (int x) {
    return x*x*x; }
```

tijelo funkcije

Tip funkcije

Tip funkcije određuje kakvog će tipa biti podatak koji funkcija vraća kôdu koji ju je pozvao:

```
int funkcijaPrva(int x);          //tipa int
float funkcijaDruga(float x);     //tipa float
char* funkPok();                  //vraća pokazivač na znak
```

return

Naredbom 'return' rezultat funkcije se proglašava povratnom vrijednošću, izvođenje funkcije se prekida, a vrijednost se vraća pozivajućem kôdu:

```
float apsolutno(float X){
   return(x>=0) ? x:-x;
}
```

*

Funkcije tipa *void*

Kad funkcija ne treba vratiti vrijednost, deklarira se tipom 'void':

```
void ispisiKvadrat(float x){
  cout<<(x*x)<<endl;}
```

Poziv funkcije

```
#include<iostream>
using namespace std;
int Funkcija(int Formalni_x);                          //deklaracija
int main(){
int Stvarni_x=2;
   int xNaCetvrtu=Funkcija(Stvarni_x)*Stvarni_x;//poziv
   int xNaDevetu=Funkcija(Funkcija(Stvarni_x)); //poziv
   int a=3;
   int b=4;
   int Suma=Funkcija(a)+Funkcija(b);                   //poziv
   cout << Suma << endl;
   system("pause");
   return 0;
}
int Funkcija(int Formalni_x){                          //definicija
   return(Formalni_x*Formalni_x*Formalni_x);
}
```

*

Program: blagajna (izdavanje računa)

```
#include<iostream>
#include<iomanip>
using namespace std;
double Total(int P1, float P2); //deklaracija funkcije
int main(){
   cout<<"Unesi cijenu <dec.podatak> i broj komada
         <cijeli broj>: ";
   int Broj;
   float Cijena;
   cin>>Cijena>>Broj; //ulancano citanje
   cout<<endl<<Broj<<" proizvoda po KM"<<setw(6)<<Cijena
         <<endl;
   cout<<"iznosi ukupno KM"<<setw(6)<<Total(Broj,Cijena)
         <<endl; //ispis s pozivom funkcije
   cout<<"ukljucujuci 22% PDV-a"<<endl;
   system("pause"); return 0;}
double Total(int P1,float P2){ //definicija funkcije
   //vraca umnozak cijene P1 i kolicine P2
   //uvecan za 22% PDV-a
   return (P1*P2+(P1*P2*.22));};
```

Funkcije: definicija - poziv

Definicija

```
int Funkcija (int x) {
    return x*x*x; }
```

formalni argument

tijelo funkcije

stvarni argument

Poziv

```
cout<< ... <<Funkcija(2)<<endl;
cout<< ... <<Funkcija(3)<<endl;
```

Ako je definicija funkcije navedena prije njenog prvog poziva u programu, nije potrebno posebno navoditi deklaraciju, već se definicija smatra ujedno i deklaracijom.

Poziv funkcije

```
#include<iostream>
using namespace std;
int Funkcija(int Formalni_x){                    //definicija
   return(Formalni_x*Formalni_x*Formalni_x);
}

int main(){
int Stvarni_x=2;
   int xNaCetvrtu=Funkcija(Stvarni_x)*Stvarni_x;//poziv
   int xNaDevetu=Funkcija(Funkcija(Stvarni_x)); //poziv
   int a=3;
   int b=4;
   int Suma=Funkcija(a)+Funkcija(b);             //poziv
   cout << Suma << endl;
   system("pause");
   return 0;
}
```

Primjer

```
#include<iostream>
using namespace std;
void Funkcija(int n){
   cout << "Funkcija: " << endl;
   while (n>0) {
       cout << n << endl;
       n/=2;
   }
}

int main(){
   int n;
   cout << "Upisite n: ";
    cin >> n;
    Funkcija(n);
   cout << "Glavni program: " << endl << n << endl;
    system("pause");
    return 0;
}
```

Argument funkcije MOŽE se mijenjati unutar funkcije

Promjene argumenta funkcije NEĆE se reflektirati izvan funkcije

1. Seta (izumitelj saha) je trazio nagradu da mu se na prvo polje sahovske ploce stavi jedno zrno psenice na drugo 2, na trece 4..., tj. na svako sljedece polje dva puta veca kolicina psenice ukupno ima 64 polja.

2. Ispisat prirodne brojeve u vidu spirale za učitani broj kolona.

3. Program za razlaganje cijelog broja na proste faktore

???

Definicija rekurzije

- **U matematici i računarstvu, rekurzija je pristup u kojem se neki pojam, objekat ili funkcija definiše na osnovu jednog ili više osnovnih (baznih) slučajeva i na osnovu pravila koja složene slučajeve svode na jednostavnije.**

- **Rekurzivna funkcija = funkcija koja poziva samu sebe, svodeći rješavanje složenog problema na jednostavniji problem iste prirode, sve dok se problem ne pojednostavi do osnovnog (trivijalnog) slučaja.**

Primjer: **Rekurzivna (induktivna) definicija** x^n

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

osnovni (bazni) slučaj

rekurzivni korak

Postoje dvije vrste rekurzije:

- **direktna rekurzija,**
- **indirektna rekurzija.**

Direktna rekurzija nastaje kad se u definiciji funkcije poziva ta ista funkcija.

Indirektna rekurzija nastaje kad jedna funkcija poziva neku drugu funkciju, a ova opet poziva funkciju iz koje je pozvana.

Definicija rekurzije

- **Osnovni elementi rekurzije:**
  - **osnovni (bazni) slučaj** = jednostavan (trivijalan) problem koji može da se riješi bez rekurzivnog poziva i koji omogućava zaustavljanje rekurzije.
  - **rekurzivni korak** = mehanizam za pojednostavljenje složenog problema, tj. svođenje složenog problema na rješavanje jednostavnijeg problema iste prirode

    Rješenje problema u *n*-tom koraku bazira se na rješenju iz (*n*-1)-og koraka.

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

osnovni (bazni) slučaj

rekurzivni korak

- **Izostavljanje osnovnog slučaja ili rekurzivnog koraka čini definiciju nekompletnom.**

Kao jednostavan primjer može se rekurzivno definisati funkcija koja opisuje faktorijel.
Izraz je:

$$n! = n * (n-1) * (n-2) * .... * 3 * 2 * 1$$

Ako se bolje posmatra gornji izraz, vidljivo je da se članovi mogu grupisati na sljedeći način:

$$n! = n * [(n-1) * (n-2) * .... * 3 * 2 * 1]$$

Time se zapravo dobije potpuno novi izraz:

$$n! = n * (n-1)!$$

Ako se sada promotri taj izraz, mogu se uočiti dva spomenuta dijela – osnovni dio (n!=1 za n=0,1) i rekurzivni dio

(n! = n * (n-1)! za n>1).

Karakteristike rekurzivnih funkcije

- kod takvih funkcija mora postojati uslov prekida kad program zadanu operaciju bude izvršavao bez pozivanja samog sebe.

Ukoliko se pokušaju izračunati vrijednosti gore navedene funkcije za nekoliko brojeva, dobije se:

$$3! = 3 * 2 * 1 = 3 * 2!$$

$$2! = 2 * 1!$$

$$1! = 1$$

Iz ovoga proizlazi da rekurzivni pozivi prestaju onda kad argument funkcije postane jednak 1, tačnije, budući da je i 0! jednako 1, može se reći da rekurzivni pozivi prestaju za 0 ili 1.

Definicija rekurzije

- **Implementacija rekurzije:**

$$x^n = \begin{cases} 1, & n = 0 \\ x \cdot x^{n-1}, & n > 0 \end{cases}$$

osnovni (bazni) slučaj

rekurzivni korak

```
double stepenovanje(double x, int n)
{
    if (n==0)
        return 1;
    else
        return x * stepenovanje(x,n-1);
}
```

osnovni (bazni) slučaj

rekurzivni korak

Definicija rekurzije

- **Analiza izvršavanja rekurzivne funkcije:**

```
double stepenovanje(double x, int n)
{
    if (n==0) return 1;
    else return x * stepenovanje(x,n-1);
}
```

stepenovanje(3,2)

x=3
n=2 ⇒ return 3 * stepenovanje(3,1)

x=3
n=1 ⇒ return 3 * stepenovanje(3,0)

x=3
n=0 ⇒ return 1

3 * 1 = 3

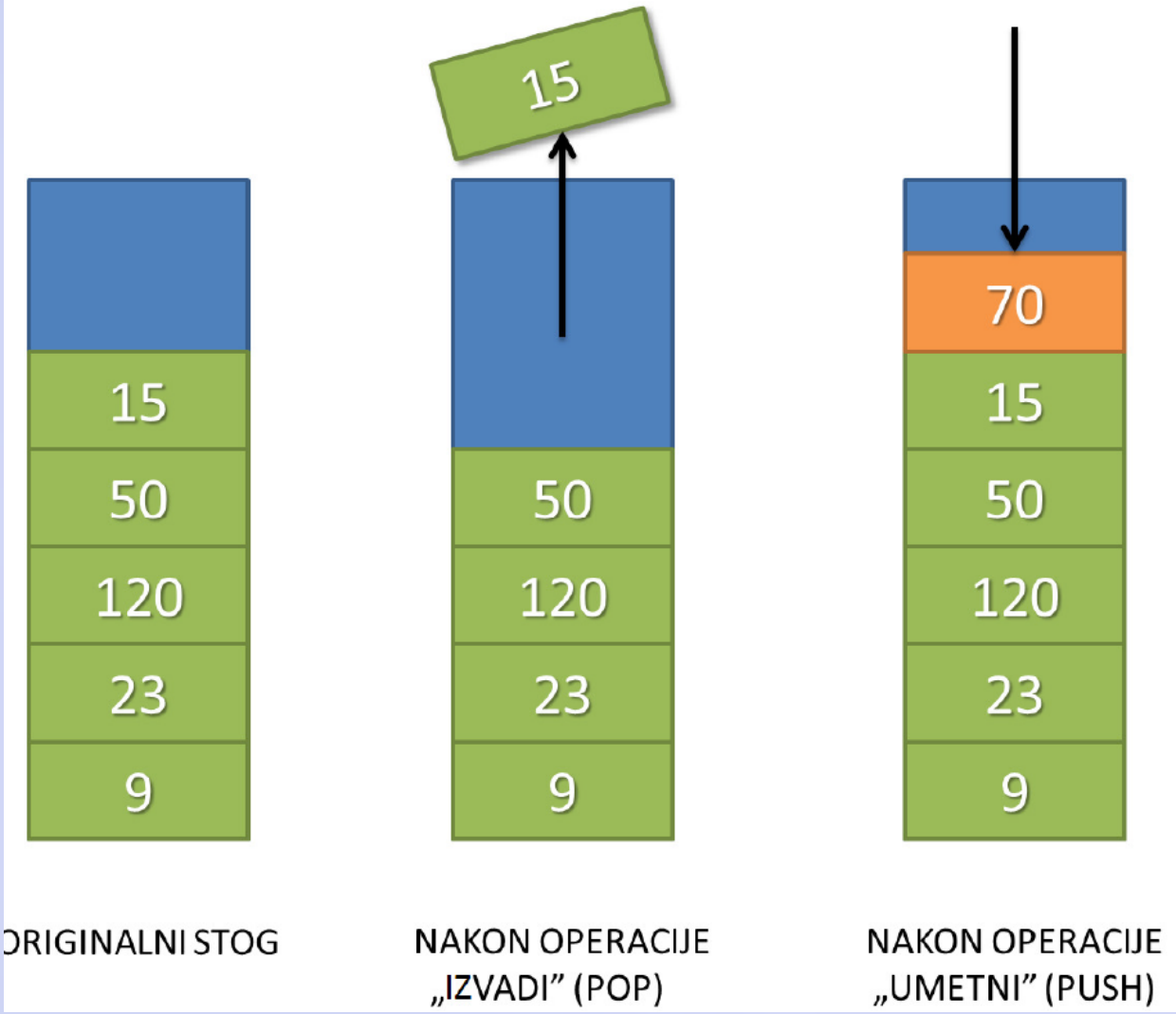
3 * 3 = 9

- **Jedan od glavnih nedostataka rekurzivnih funkcija jest to što takve funkcije zauzimaju više memorijskog prostora, a uglavnom i zahtijevaju više vremena za izvođenje od odgovarajuće iterativne funkcije.**

Kako rade rekurzivni programi?

Kod rada s rekurzivnim funkcijama koristi se struktura podataka koja se naziva stog (engl. stack) u koju se pohranjuju varijable rekurzivne funkcije.

No, što je to zapravo stog? Stog je apstraktni tip podataka koji služi za pohranu niza istovrsnih elemenata. Budući da stog radi po LIFO principu (engl. Last In First Out), posljednji pohranjeni podatak se uzima prvi u obradu.


15
15
50
120
23
9
50
120
23
9
70
15
50
120
23
9
ORIGINALNI STOG
NAKON OPERACIJE „IZVADI” (POP)
NAKON OPERACIJE „UMETNI” (PUSH)

Definicija rekurzije

Primjer:

```
#include <iostream>
Using namespace std;
double stepenovanje(double x, int n)
{
   if (n==0)
      return 1;
   else
      return x*stepenovanje(x,n-1);
}

int main()
{
   double x;
   cout<<"x=";
   cin>>x;
   for (int n=0; n<5; n++)
      cout<<x<<„^“<<n<<stepenovanje(x,n));
cin>>x;
   for (int m=10; n<20; n++)
      cout<<y<<„^“<<m<<stepenovanje(y,m));

   return 0;
}
```

Primjer izvršavanja:

```
x=3
3.00^0=1.0000
3.00^1=3.0000
3.00^2=9.0000
3.00^3=27.0000
3.00^4=81.0000
```

Princip realizacije rekurzivne funkcije je sljedeći:

- **rekurzivna funkcija** poziva se sa određenom početnom vrijednošću varijable;
- računski proces unutar funkcije odvija se nesmetano sve do tačke u kojoj dolazi do poziva istog potprograma sa izmijenjenom (najčešće smanjenom) vrijednošću argumenta;
- novi poziv iste funkcije dovodi do privremenog prekida izvršavanja tekućeg poziva rekurzivne funkcije. Pri tome se sve vrijednosti lokalnih varijabli spremaju na stog;
- ponavljanjem opisanog procesa stalno raste sadržaj stoga, sve dok ne stigne do krajnjeg uslova koji više ne zahtijeva rekurzivne pozive funkcije;
- povratkom s nižeg nivoa izvršavanja rekurzivne funkcije na viši nivo nastavlja se računski proces koji je bio prekinut, učitavaju se i obnavljaju vrijednosti lokalnih varijabli koje su bile pohranjene u stog;

– kad je dobiven pretposljednji rezultat rekurzivne funkcije, stog je doveden u početno stanje, izračunava se krajnja vrijednost rezultata i dostavlja pozivajućem programu.

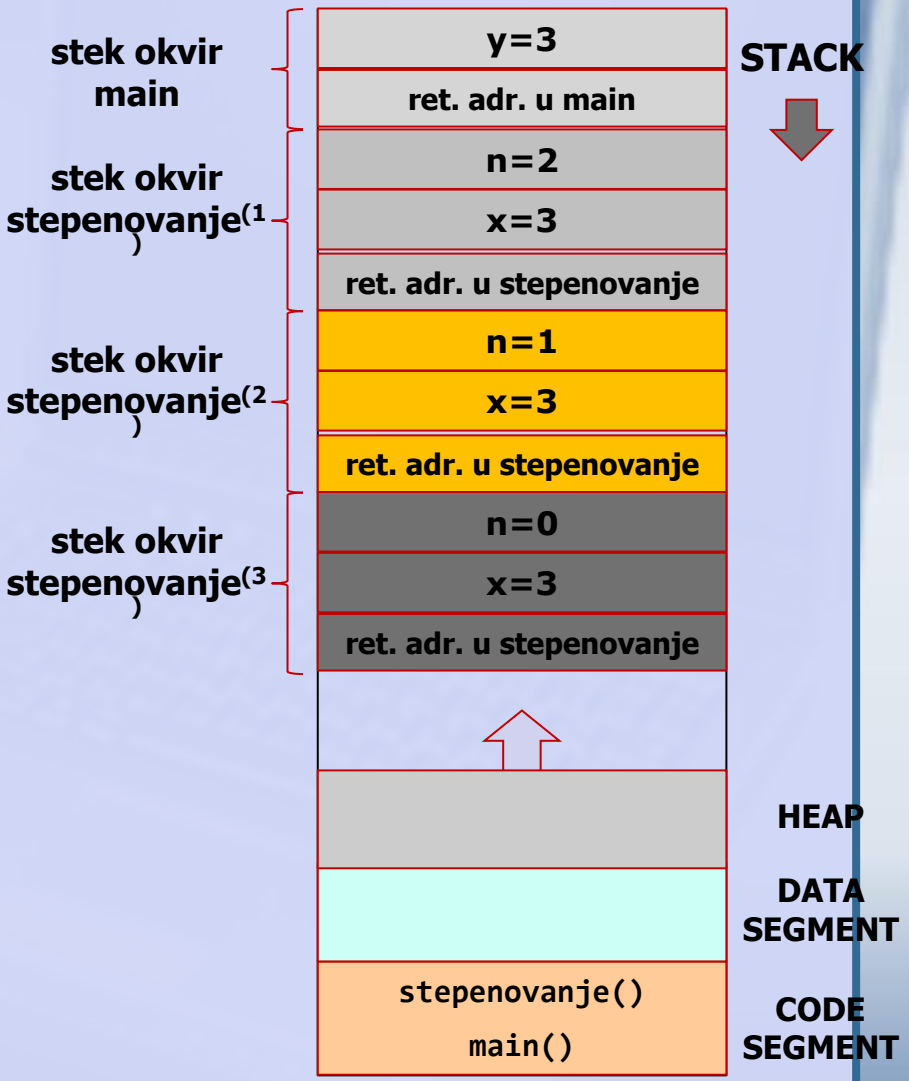
Budući da je dokazano da se svako rekurzivno rješenje može napisati kao iterativno, rekurzija se preporuča koristiti samo onda kad se iterativnim pristupom teško dolazi do rješenja problema.

Proces izvršavanja rekurzije

Primjer:

```
#include <iostream>
using namespace std;
double stepenovanje(double x, int n)
{
   if (n==0)
      return 1;
   else
      return x*stepenovanje(x,n-1);
}

int main()
{
   double y;
   cout<<"y=";
   cin>>y;
   cout<<y<<stepenovanje(y,2));
   return 0;
}
```



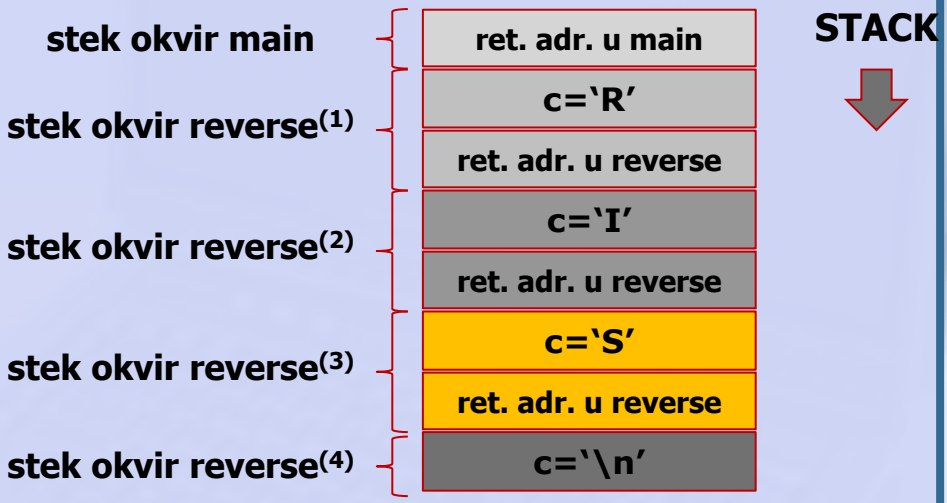
Proces izvršavanja rekurzije

Primjer:

```
#include <iostream>
using namespace std;
void reverse()
{
   char c;
   cin>>c;
   if (c != '\n')
   {
      reverse();
      cout<<c;
   }
   return;
}

int main()
{
   reverse();
   return 0;
}
```

Po povratku iz pozvane funkcije (završen rekurzivni korak), nastavlja se izvršavanje od mjesta na kojem je prekinuto izvršavanje.



Primjer izvršavanja:

```
RIS
SIR
```

Zašto rekurzija mora da konvergira?

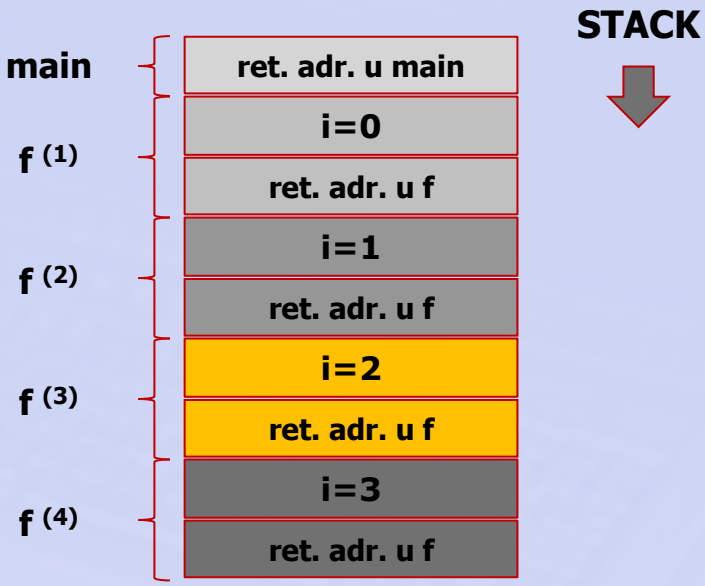
Primjer 1:

```
#include <iostream>
using namespace std;

void f(int i)
{
   f(i+1);
   return;
}

int main()
{
   f(0);
   return 0;
}
```

Svaka nova instanca pozvane funkcije ima svoj stek okvir, a veličina steka je ograničena!



Primjer 2:

```
int bad(int n)
{
   if (n == 0) return 0;
   return bad(n/3 + 1);
}
```

bad(1)

n=1 ⇒ return bad(1)

n=1 ⇒ return bad(1) ???

Poređenje rekurzivnih i iterativnih programa

rekurzivni proračun sume	iterativni proračun sume

`int suma (int n){`
`  if (n == 1)`
`  return 1;`
`  else`
`  return suma(n - 1) + n;`
`}`
	`int suma (int n){`
	`  int sum=0;`
	`  while (n > 0){`
	`  sum += n;`
	`  n--;`
	`  }`
	`  return sum;`
	`}`

- iako je rekurzivni kod kraći i „elegantniji“, isto tako je i neučinkovitiji (sporiji) te teži za analiziranje, no i pored toga preporučuje se korištenje rekurzivnih funkcija u onim slučajevima kad to odgovara prirodi problema.

Dobre i loše strane rekurzije

Dobre strane rekurzije

Kod je (obično):

- kratak, čitljiv i jednostavan za razumijevanje,
- jednostavan za održavanje i otklanjanje grešaka,
- pogodan za dokazivanje korektnosti,
- ...

Rekurziju treba koristiti:

- ako je rekurzivno rješenje "prirodno" i jednostavno za razumijevanje,
- ako rekurzivno rješenje ne zahtijeva suvišna izračunavanja koja je teško eliminisati,
- ako je ekvivalentno iterativno (nerekurzivno) rješenje previše kompleksno.

Loše strane rekurzije

- Cijena poziva:
  - svaki rekurzivni korak znači novi stek okvir i kopiranje argumenata na stek, što dalje znači novo memorijsko zauzeće i usporavanje izvršavanja
  - u slučaju "dubokih" rekurzija, prostorna (memorijska) i vremenska složenost mogu biti kritične
- Suvišna izračunavanja:
  - svođenje složenog problema na jednostavnije može da rezultuje suvišnim ponavljanjima istih izračunavanja